计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210500104

# 前向纠错编码在网络传输协议中的应用综述

林利祥[1]　刘旭东[1]　刘少腾[2]　徐跃东[1]
1 复旦大学信息科学与工程学院　上海 200433
2 华为技术有限公司网络技术实验室　广东 深圳 518129
(lxlin19@fudan.edu.cn)

**摘　要**　前向纠错编码是一种在网络传输中应对丢包的技术。在传输过程中加入冗余数据,使接收端在丢包场景下可通过冗余数据直接恢复出原始数据。在丢包多、时延大的场景下,适当加入前向纠错编码可以大量节省超时重传的等待时间,从而提高网络传输的服务质量。过多地添加冗余会造成带宽的浪费,而过少地添加冗余会导致服务器端接收到的数据不足以恢复在传输过程中丢失的数据,因此实际应用前向纠错编码的难点在于恰当地控制冗余数据的比例。目前,前向纠错编码研究大多基于传统网络协议。而随着QUIC(Quick UDP Internet Connections)协议的崛起,由于其具有0-RTT(Round Trip Time)连接、多路复用以及连接无缝迁移等特性,因此更多的前向纠错研究开始结合QUIC协议,以进一步提升传输性能。文中首先对前向纠错编码(Forward Error Correction Coding,FEC)进行了概述,介绍了其应用场景、基本策略和自适应冗余控制策略;然后详细介绍了单播场景和组播场景中,前向纠错编码在传统协议中的研究现状;最后介绍了前向纠错编码在QUIC协议中的研究现状及仍面临的挑战。

**关键词**:QUIC协议;TCP协议;前向纠错编码;自适应冗余度;性能测量
**中图法分类号**　TP393

## Survey on the Application of Forward Error Correction Coding in Network Transmission Protocols

LIN Li-xiang[1],LIU Xu-dong[1],LIU Shao-teng[2] and XU Yue-dong[1]
1 School of Information Science and Engineering,Fudan University,Shanghai 200433,China
2 Network Technology Laboratory Huawei Technologies Co., Ltd.,Shenzhen,Guangdong 518129,China

**Abstract**　Forward error correction (FEC) coding is a technique to cope with packet loss in network transmission. By adding redundant data to the transmission process,the receiver can recover the original data directly from the redundant data in packet loss scenarios. In the scenario of high packet loss and high latency,adding forward error correction coding appropriately can save a lot of waiting time for timeout retransmission to improve the quality of service of network transmission. Adding too much redundancy will result in bandwidth wasting,while insufficient redundancy will fail to recover lost data,so the difficulty of using forward error correction coding in practice is to properly control the proportion of redundant data. Nowadays,most of the forward error correction coding research is based on traditional network protocols,but with the rise of QUIC (quick UDP internet connections) protocol,more forward error correction researches start to incorporate QUIC protocol due to its characters like 0-RTT (round trip time) connectivity,multiplexing,seamless connection migration to further improve transmission performance. This paper gives an overview of forward error correction coding,introduces its application scenarios,basic policies and adaptive redundancy control strategies. Then it introduces the research status of forward error correcting coding in traditional protocols in unicast and multicast scenarios. Finally,this paper introduces the current research status and challenges of forward error correction coding in QUIC protocol.

**Keywords**　QUIC protocol,TCP protocol,Forward error correction coding,Adaptive redundancy,Performance measurement

## 1　引言

前向纠错编码是一种在网络传输过程中控制丢包率的技术,其通过冗余数据对丢失的原始数据进行恢复,以降低丢包率。相比等待超时之后自动请求重传(Automatic Repeat-reQuest,ARQ),前向纠错编码可以直接依赖冗余数据对数据

到稿日期:2021-05-14　返修日期:2021-10-15
基金项目:广东省重点领域研发计划(2020B010166003);国家自然科学基金(61772139)
This work was supported by the Guangdong Provincial Key R&D Programme(2020B010166003) and National Natural Science Foundation of China(61772139).
通信作者:徐跃东(ydxu@fudan.edu.cn)

进行恢复，而不需要等待重传，为数据传输的实时性提供了有力的保障。

如今的前向纠错编码已服务于计算机网络的各种场景及应用，如组播、视频流媒体、文件传输等。在这些场景中，前向纠错编码呈现出很多明显的优势。例如，对组播场景而言，为发送的数据添加前向纠错编码可以有效地保护数据，大大减少超时重传的风险，对组播的传输效率有显著地提升。在用户对数据实时性要求较高的场景和应用中，例如，用户观看视频流媒体时，合理地应用前向纠错编码可以减少直播的延迟以及视频的卡顿，从而大幅提升用户体验质量（Quality of Experience，QoE）。

如今的前向纠错编码研究大多基于传统网络协议。而随着近年来谷歌公司提出 QUIC 协议[1]，原本很多结合传统协议的前向纠错编码研究开始结合 QUIC 协议进行研究。相比传统的网络协议，QUIC 协议具有很多优势：首先，QUIC 协议的 0-RTT 连接特性相比传统协议，如传输控制协议（Transmission Control Protocol，TCP）的 3 次握手，建立连接更快，多路复用特性使 QUIC 可以有效应对队头拥塞问题，解决重传模糊问题，对数据包进行更精准的控制，其无缝迁移特性使 QUIC 上层业务逻辑不易断开。其次，QUIC 是基于 UDP 的应用层协议，每次更新与修改都不需要重新部署系统内核，相比内核态的 TCP 协议，QUIC 更新周期更短、修改更便捷。因此，基于 QUIC 协议本身的这些优势，更多的前向纠错编码研究开始结合 QUIC 协议，期望能在各种不同的网络场景（如单播、组播）及不同的网络应用（如文件传输、流媒体传输等）中获得更大的性能提升。

本文首先对前向纠错编码技术进行了概述，包括前向纠错编码的应用场景、基本策略和自适应控制策略；然后详细介绍了前向纠错编码结合传统协议的研究现状，讲述了单播场景和组播场景中各项研究的动机、建模方法以及实验效果；最后对新兴的 QUIC 协议以及前向纠错编码结合 QUIC 协议的研究现状进行了介绍，并提出了前向纠错编码目前仍面临的种种挑战。

## 2 前向纠错编码概述

本节首先介绍了前向纠错编码的应用场景，随后介绍了前向纠错编码的基本策略，主要包括分组码与卷积码，最后对自适应冗余度选择策略的研究现状进行了介绍，如图 1 所示。

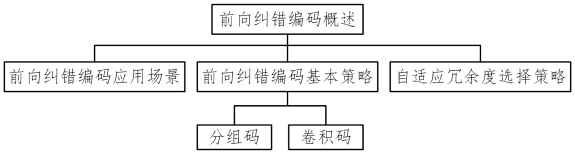



图 1 前向纠错概述部分

Fig. 1 Summary part of forward error correction

### 2.1 前向纠错编码应用场景

在现实生活中，高延迟、高丢包的复杂网络场景不胜枚举：如高铁上的移动设备用户频繁进行基站切换、无人机高速运动、远距离跨国视频连线等。在这种存在大量丢包且用户对数据实时性要求较高的场景中，数据传输的延迟以及数据包丢失造成的卡顿将大大降低网络服务质量（Quality of Service，QoS），因此保证数据的实时性成为了新的挑战。

传统应对丢包问题的策略依赖于数据链路层的自动请求重传协议，然而等待超时后再请求数据包重传的过程在一些场景下比较低效，往往不能满足用户对数据实时性的要求。在这种背景下，前向纠错算法是一种较为理想的解决方案。前向纠错编码通过在数据包中加入冗余，可以让数据接收端在丢包情况下，直接从冗余中恢复丢失数据，而不需要发送端重传。这对数据传输的实时性有了很大的提升[2-3]，在对数据实时性要求较高的场景下具有非常重要的作用。

目前，对于前向纠错编码的研究主要聚焦于两方面，分别是编码的策略研究与自适应前向纠错冗余度控制算法的研究。

### 2.2 前向纠错编码策略

编码策略的本身是期望通过设计前向纠错编码算法来提高恢复丢包的效率。前向纠错编码策略主要分为分组码与卷积码。

#### 2.2.1 分组码

分组码在网络前向纠错编码中将数据包分成不同的组（block），每组数据独立进行编码。例如，将原始数据包中每 $k$ 个数据包分为一组，每组中添加 $r$ 个数据包，从而使分组长度变为 $n$。

我们通常用符号 $(n,k)$ 表示分组，$n$ 表示分组中的数据包总数，$k$ 表示码组中原始数据包的个数，$r=n-k$ 则表示冗余数据包的个数。对于分组码而言，每个分组中最多可容许 $r$ 个数据包丢失。超过 $r$ 个数据包丢失将无法通过分组内的数据进行恢复。

XOR 编码是一种简单、原始的分组码，它的策略是将 $k$ 个数据包分为一组，在 $k$ 个数据包的最后添加 1 个冗余数据包，该数据包由分组中的第一个数据包按顺序与其后的数据包进行异或（XOR）操作求得。若 $k+1$ 个数据包中丢失了 1 个数据包，则可以通过自后向前的异或操作恢复丢失的数据包。XOR 编码虽然逻辑简单，实现方便，但只能为一个分组添加一个冗余数据包，这限制了 XOR 算法的使用。

RS（Reed-Solomon）码[4-5]于 1960 年被提出，是一种经典的分组码。RS 码主要利用范特蒙矩阵或柯西矩阵的特性，对 $k$ 个数据包进行分组，并根据 $k$ 个数据包生成 $r$ 个冗余数据包，$k$ 个数据包和 $r$ 个冗余数据包组成一个长度为 $n$ 的数据包的分组。在一个分组中，小于等于 $r$ 个数据包的丢失可以通过其他数据包恢复，而超过 $r$ 个数据包的丢失则无法恢复。目前国际互联网工程任务组（The Internet Engineering Task Force，IETF）对 RS 码已经明确了添加冗余数据的执行规范[6]。

#### 2.2.2 卷积码

卷积码最早由 Elias 于 1955 年提出。其生成方式是将待传输的信息序列通过移位寄存器进行编码。它与分组码的主要区别在于：卷积码有记忆编码，即编码器的 $n$ 个输入不仅与当前时段的 $k$ 个输入有关，还与存储器之前的 $m$ 个输入有关。一般卷积码被表示为 $(n,k,m)$ 编码，表示编码中的 $n$ 个输出、$k$ 个输入和 $m$ 个移位寄存器。为了增强纠错能力，

卷积码通过卷积引入了时间相关性，使得编码结果与之前的输入有关。如今，卷积码一般采用的译码方法是 Viterbi[7] 于 1967 年提出的最大似然译码法。IETF 也明确了针对 RLC 算法添加冗余数据的执行方法[8]。

2017 年 Roca 等[9]在第三代合作伙伴计划(3GPP)多媒体组播(Multimedia Broadcast/Multicast Service，MBMS)环境中对 RS 码与卷积码的性能进行对比。他们在仿真环境 OpenFEC 下模拟 3GPP 标准的信道进行测试发现，卷积码相比分组码解码延时更短。

### 2.3 自适应冗余度选择策略

上述编码策略考虑的是编码策略本身。然而事实上，真实的网络环境非常复杂多变。在传输过程中添加不必要的冗余会造成带宽无谓的浪费，添加冗余过少，则不能恢复丢失的数据包，同样造成带宽浪费。要确定每份数据中添加的冗余数据的量，需要对网络状态做到即时、比较精准的预测与评估。

2000 年，Padhye 等[10]率先提出自适应编码的概念，即根据网络环境的不同，动态地调整每份数据中的冗余量，以尽量保证冗余数据包被充分利用，不浪费带宽。

## 3 前向纠错编码结合传统协议发展现状

本节介绍传统协议结合前向纠错编码的研究现状，分两个场景进行介绍，分别为单播场景下与组播场景下前向纠错技术结合传统协议的研究现状。

### 3.1 单播场景下传统协议结合前向纠错编码的发展现状

早在 2002 年，Barakat 等[2]就研究了如何在 TCP 中平衡 FEC 数据量的问题。他们的研究表明：在丢包环境中适当增加冗余数据包可以增加网络通信的吞吐量。然而添加冗余数据包的比例存在一个最优值，超过这个最优值之后继续添加的冗余数据包会使吞吐量下降。另外，如果链路中平均丢包率不变，对于分组码而言，若不增加冗余数据包的冗余度，增加分组的长度也可以使数据更好地应对突发性丢包(burst loss)。

2004 年，Baldantoni 等[11]针对无线环境提出了一种新的自适应冗余添加策略并选择了分组码进行实验。在算法方面，他们参考了文献[12]中的“平方根公式”，将吞吐量估计为：

$$T=\min\left\{\frac{1}{RTT}\sqrt{\frac{3}{2p}},\mu\right\}$$

其中，$T$ 表示吞吐量，$p$ 是数据包发送的丢失概率，$\mu$ 是瓶颈带宽。随后经过一系列的推导，将应添加的冗余 $R$ 表示为：

$$R=\mu * K * RTT\sqrt{\frac{2P_{N,K}}{3}}-K$$

其中，$K$ 是分组数据包个数，$P_{N,K}$ 表示丢失一个 TCP 包又无法恢复的概率。

实验在 NS2 仿真环境的 Gilbert-Elliot 模型上进行，结果表明，自适应添加冗余的 TCP 相比默认的 TCP 以及 NewReno 等拥塞控制算法吞吐量更高。然而相比 Westwood 等针对无线网络的 TCP 拥塞控制策略，该策略的性能表现与其还存在一定差距。

2004 年，Lundqvist 等[13]提出：在端到端 TCP 传输中使用 FEC 编码时，编码分组的大小应小于当前拥塞控制窗口(win_size)的大小，最好保留一些宽裕度。当发送窗口很小时，最好禁用 FEC 功能，因为这会增加丢包的风险。他们还提出了自适应的 FEC 算法，其逻辑为：1)如果当前 win_size$<L1$(实验中设 $L1$ 为 10 个数据包)，则关闭 FEC 策略；2)如果当前 $L1<$win_size$<L2$(实验中设 $L2$ 为 40 个数据包)，则使分组长度(block_size)为当前 win_size 的一半；3)如果当前 win_size$>L2$，则使 block_size 为设定的最大值(maximum value)。在实际操作中，Lundqvist 等为 FEC 更新策略添加了更新滞后条件，以避免 FEC 开关过于频繁。在拥塞控制窗口大小减少到 8 个包后禁用 FEC 策略，并在拥塞控制窗口大小增加到 12 个包以上后再次启用。这样操作，发送冗余信息的比例大约在 5%～25%之间。实验在 NS2 仿真平台上进行，结果表明，与不添加 FEC 冗余包的 TCP 相比，在丢包和延时较大时，利用自适应 FEC 策略可以有效提高吞吐量。

2005 年，Ahn 等[14]针对有损的无线信道环境(802.11 标准)，提出了 AFECCC 自适应 FEC 算法：如果在一个初始化的定时器(Drop Timer，DT)期间所有的数据包都成功传输，那么冗余级别就降低一级；如果检测到丢包，那么 RS 码冗余级别就会立即上升一级，并且不断利用参数 $\alpha$ 与 $\beta$ 对定时器时长进行修改。实验在仿真环境与真实环境中进行，并比较了不同误码率(bit error rate)下的传输性能比(performance ratio)。结果表明，与 ARQ 结合 FEC 冗余控制算法(LA-IR 及 LA-IR II[15])及固定冗余比的 RS 码相比，AFECCC 算法的性能最好。

Tsugawa 等[16]于 2007 年指出，AFECCC 算法效果很依赖于定时器的更新公式中 $\alpha$，$\beta$ 两个参数的设置，且该算法与 TCP 直接结合效果不佳。例如，TCP 拥塞控制算法在未丢包的情况下不断地扩大拥塞控制窗来试探带宽，在这个过程中，AFECCC 的冗余级别不断降低，当拥塞控制窗过大超过网络瓶颈带宽时，冗余级别应该足够高，以保证丢包的恢复，然而这时 AFECCC 的冗余级别却很低。他们针对流媒体(video streaming)场景，基于 AFECCC 提出了一个新的自适应 FEC 方案，称为 TCP-AFEC。其本质是在 AFECCC 算法上添加阈值作为约束，否决错误的窗长变化决策并在发生丢包时更新该阈值。若当前拥塞控制窗小于该阈值，即使在定时器期间中没有丢包，TCP-AFEC 也不会降低冗余级别，从而避免当拥塞控制窗减小时，冗余度变得过低。实验在 NS2 仿真环境下搭建拓扑进行，比较参数为不同的视频码率下的数据到达比。实验结果表明：与固定冗余度的静态(static)FEC 算法、TCP-SACK 及 TCP-AV 相比，TCP-AFEC 在数据到达比方面性能最佳；在固定视频码率下，TCP-AFEC 并发的连接数也更多。

Convertino 等[17]于 2006 年提出，在无线网络(IEEE 802.11)的有损视频传输场景下，虽然许多视频解码器可以掩盖少量随机丢包的影响，但是面对突发性丢包(burst loss)是很难修正的。他们提出用一个 4 状态马尔可夫模型来描述数据包达到的状态。这 4 个状态为：接收到数据包；在一个突发性损失中收到数据包；在一个突发性损失中丢失了数据包；在

一个普通丢包(gap)状态中丢失了数据包。他们提出的 XR 算法的核心理念是，利用分组码中的 RS 码添加冗余数据时，数据分组要尽量覆盖突发性损失。XR 算法中首先定义了连续收到数据包个数的最小值$g_{\min}$。在传输过程中，若连续收到的数据包个数始终低于该值，便视为当前处于突发性丢包状态(burst loss)，反之视为非突发性丢包状态(gap)。Convertino 等通过获取突发性丢包状态的数据包长度(burst_len)、突发性丢包状态中丢失数据包概率(burst_density)来自适应地控制自适应 FEC 策略的分组数据包个数 $n$ 和每个分组内原始数据包个数 $k$。

$$n=burst\_len+g_{\min}$$

$$n-k=n\times burst\_density$$

实验在 NS2 仿真环境下进行，模拟了 802.11 无线环境下不同的丢包模式，与平均丢包率模型(Basic Packet Loss Rate，BasicPLR)相比，XR 算法能较好地应对突发损失，优于基于平均 PER 的自适应 FEC 调整方法。他们引入了 FEC 效率(FEC Efficiency，FE)来评估算法性能，FE 定义为：

$$FE=1-\frac{1-RC}{1-FECredundancy}$$

其中，$RC$(Recovered Capacity)表示恢复数据占总丢失数据的百分比，$FECredundancy$ 表示 FEC 数据包的冗余比。实验结果表明，在绝大多数环境下 XR 算法的 FE 比其他算法更高。

Flach 等于 2013 年提出 TCP-IR 的执行方案[18]，他们将 FEC(XOR)策略集成到 TCP 中，以固定的间隔插入 XOR 的冗余数据包，在接收端可以做到快速的数据恢复，从而避免丢包后的重传。然而这样固定地插入冗余数据包在变化的网络环境中显然不够合理。

Nagy 等[19]于 2014 年提出利用冗余数据包结合 RTP 协议传输视频数据，并且提到冗余数据包不仅可用于错误恢复，本身也可以用于拥塞控制。他们将控制逻辑分为保持(STAY)、探测(PROBE)、冗余度提升(UP)和冗余度下降(DOWN)。若在 STAY 状态下未产生丢包，则系统会进入 PROBE 状态，发送端尝试对数据添加冗余数据包，以进行网络的带宽探测；如果添加冗余包后数据包发生了丢失，就尝试用冗余数据包进行恢复，并在下次传输数据时继续保持原有的状态(STAY)；如果添加了冗余数据包且未发生丢包，则发送端进入 UP 状态，将视频传输码率提高；若 STAY 状态下还存在丢包，则进入 DOWN 状态。他们创新性地提出了一种新的拥塞控制算法 FBRA，并与其他两种拥塞控制算法(RRTCC 与 C-NADU)进行了性能比较。实验在 NS2 仿真平台与真实环境下进行，结果表明，RRTCC 和 C-NADU 在改善带宽利用率和丢包率上各有优势。而 FBRA 算法兼顾了改善带宽利用率及丢包率，它的吞吐量比 C-NADU 更大，丢包率比 RRTCC 更低。尽管它们的吞吐量相近，但 FBRA 为用户提供了更好的用户体验，因为它的包延迟变化和吞吐量变化都相对较低。

2018 年，Ferlin 等[20]提出了基于 XOR 算法的动态 FEC 算法(TCP-dFEC)，该算法原型为 TCP-IR，相比 TCP-IR，该算法实现了一个动态的 FEC 控制机制。另外他们将 TCP-dFEC 拓展为多路径(MultiPath，MP)的 TCP-dFEC，并在 MPTCP 环境下进行部署，最终在 Linux 内核中得以实现。TCP-dFEC 的核心思想就是控制残差(residual)，并根据残差来修改冗余比(ratio)。冗余比表示间隔多少个包加一个冗余数据包，文中用 $T$ 表示一个时间周期(interval)，并用 $N$ 表示时间周期的序号。

$$Residual_i=\frac{Retransimit}{Total-Retransmit}$$

$$\overline{Residual}=\frac{\sum_{n=1}^{N}Residual_n}{N}$$

实验中，设置 $\delta=0.33$，$N=2$ 并限制 $ratio\geqslant 4$ 进行冗余度的更新，参数 $target$ 自行设置，表示目标残差。

if $\overline{Residual}>target$ then

$$ratio'=ratio\times(1-\delta)$$

else

$$ratio'=ratio\times(1+\delta)$$

end if

实验结果表明，与传统 TCP 及 TCP-IR 相比，新算法有效缩短了超文本传输协议 2.0(Hypertext Transfer Protocol，HTTP/2)的网页下载时间并提高了 H.264 格式视频的帧率。

为了更好地利用前后状态的关联性，Cheng 等[21]于 2020 年提出了 DeepRS 自适应冗余控制算法，其神经网络模型为长短期记忆(Long Short Term Memory，LSTM)网络，通过预测包丢失的概率来自适应地调整 RS 编码器的冗余比。LSTM 网络在文献[22]中被首次提出，常用于处理序列化的数据。实验中，网络的输入为历史的包丢失模式，输出为预测的包丢失模式。仿真实验模型为 Gilbert-Elliot 信道，并利用真实的数据集[23]进行仿真环境设计。实验将算法与多个固定冗余度的 FEC 添加策略进行对比，并以恢复比(recovery ratio)作为性能衡量指标。实验结果表明，相比固定冗余度的 RS 策略，DeepRS 在实验测试中恢复比可提升至 70%。

### 3.2 组播场景下传统协议结合前向纠错编码的发展现状

在组播环境下冗余编码的研究也历时已久。2006 年，Lohmar 等[24]讨论了多媒体多播/广播服务(MBMS)功能下 FEC 功能的性能。对于多媒体多播/广播服务而言，有 3 种纠错方式：前向纠错编码、单点对单点重传纠错(Point-To-Point，PTP)、单点对多点重传纠错(Point-To-Multipoint，PTM)。他们在仿真环境中模拟多媒体组播/广播服务(MBMS)功能下，对 100000 名用户、不同大小的文件(2 MB，1 MB，512 KB)进行分发传输测试。

实验采用指数分布函数来模拟每个接收器上的数据丢失概率，结果表明，在组播环境下，相对固定的丢包状态下，冗余度的选择存在一个最优解。另外，适当添加 FEC 后，使用 PTP 结合 PTM 的修复数据方式比单纯 FEC 结合 PTP 的性能更好，可以更快地完成数据的传输。

2008 年，3GPP 建议在组播环境中使用前向纠错算法，并采用旋风码(Raptor)[25]作为纠错编码，Raptor 编码在文献[26]中被首次提出。

2010 年，Alexiou 等[27]首次对在 LTE 蜂巢网络环境下，3GPP 标准化的 Raptor-FEC 机制应用于组播数据传输

(MBMS)中的性能进行验证。Raptor 编码本质上是一种喷泉码,其恢复数据包失败的概率为:

$$p_f(m,k)=\begin{cases}1, & \text{if } m<k\\ 0.85\times 0.567^{m-k}, & \text{if } m\geqslant k\end{cases}$$

其中,$m$ 为收到的数据包的个数,$k$ 为原始数据包的个数,$p_f$ 为恢复数据包失败的概率。他们在仿真环境下针对一定丢包率的组播环境,不同组播用户人数(multicast user population)计算传输一个完整的文件的总成本。总成本是初始文件传输成本、FEC 编码导致的冗余数据包传输成本和丢失数据包的选择性重传成本的总和。实验结果表明,在有一定丢包率的组播环境中,根据丢包率适当增加冗余数据包可以有效减少组播网络中的传输成本。

Bouras 等[28]在 2012 年研究了 3GPP 标准下的旋风 Q 码(RaptorQ)的性能,并在仿真环境下与 3GPP 使用的标准旋风码(Raptor)进行对比。RaptorQ 码与 Raptor 码的编码过程几乎相同。两者的区别在于,传统 Raptor 码运行于 GF(2)域[25],新的 RaptorQ 算法运行于 GF(256)域而不是 G(2)域。在更大的有限域上进行操作,使得 RaptorQ 可以将 56403 个数据包编码到一个分组(source block)中,而 Raptor 码只能对 8192 个数据包进行操作,且 RaptorQ 可以生成多达 16777216 个编码数据包(encoding symbols),是 Raptor 码的 256 倍。同时,RaptorQ 引入了八进制运算并结合较低复杂度的 GF(2)操作,使得绝大多数符号操作都运行于 GF(2)中,只有小部分运行于 GF(256)上。引入 GF(256)使得收到 $k$ 个原始数据包与 $m$ 个冗余数据包后,恢复出原始数据包的概率为 $1-1/256^{m+1}$,而原本理想恢复概率为 $1-1/2^{m+1}$。他们在一个开源仿真平台[29]上模拟多个移动用户设备在组播环境中的情况,在带宽相同、数据冗余度相同的情况下,RaptorQ 保护的用户比例更高,即收到相同数量的冗余数据包时,其解码成功概率更高,并且为了完成所有数据的下载,Raptor 需要额外传输数据的包会更多。

## 4 前向纠错编码结合 QUIC 协议发展现状

### 4.1 QUIC 协议概述

本节对前向纠错编码结合 QUIC 协议发展现状进行了介绍,如图 2 所示。主要包括:QUIC 协议概述、QUIC 协议特性、QUIC 协议研究现状以及 QUIC 协议结合前向纠错编码发展现状。在 QUIC 协议特性部分的介绍中,我们对 QUIC 协议的 0-RTT 连接、多路复用等特性进行了介绍。在 QUIC 协议研究现状部分的介绍中,我们主要对 QUIC 协议的性能测试、协议拓展及其他研究进行了介绍。

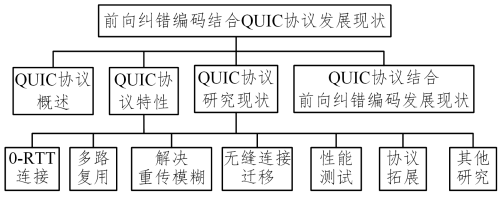



图 2 QUIC 协议概述部分

Fig. 2 Summary part of QUIC protocol

QUIC 协议是谷歌公司提出的一种基于 UDP 的全新互联网安全传输协议,最早于 2013 年发布,该版本被称为 gQUIC[1]。谷歌公司期望它能够成为安全传输层协议(Transport Layer Security,TLS)结合 TCP 的替代品,为大众提供更好的网络服务。

如图 3 所示,QUIC 协议的实现是基于 UDP 底层协议,这使得 QUIC 避免被网络端口认为是不安全的协议。相比传统 TCP 协议,QUIC 可以更快地建立数据传输的连接。如今 QUIC 已被部署在谷歌的 Chrome 浏览器以及油管(Youtube)流媒体服务上。据统计[30-31],数千个部署有 QUIC 的服务器已经处理了十亿级的网页请求,30%的谷歌流量和 7%的全球流量都使用 QUIC 协议。这得益于其应用层的实现,QUIC 的版本迭代速度相比传统 TCP 协议也有着极大提升。

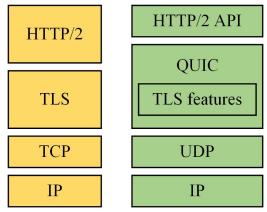



图 3 TCP 与 QUIC 协议栈

Fig. 3 TCP and QUIC protocol stack

QUIC 融合了 TCP,TLS 与 HTTP/2 的特性。虽然基于 UDP 实现,但 QUIC 和 TCP 一样有着拥塞控制、流量控制等功能,并实现了 TLS 的加密功能。与 TCP 相比,QUIC 能用更少的往返通信时间建立安全的会话。可以说,其整合了 TCP 协议的可靠性以及 UDP 协议的传输效率。

目前 QUIC 协议已经成为超文本传输协议 3.0[32](Hypertext Transfer Protocol,HTTP/3)的标准。为了更好地推广 QUIC 协议,谷歌公司已经于 2015 年将 QUIC 的标准化交给了 IETF。由 IETF 标准化的 QUIC 协议版本一般被称为 iQUIC。谷歌公司在自己的应用中使用 gQUIC,但在 gQUIC 版本更新过程中依旧会参考 iQUIC 标准。

### 4.2 QUIC 协议的特性

QUIC 协议相比传统 TCP 协议具有几个代表性的特性,包括 0-RTT 连接、多路复用、解决重传模糊和连接无缝迁移等。

#### 4.2.1 0-RTT 连接特性

对于 TCP+TLS 协议而言,建立可靠连接需要 3 个 RTT。为了保证数据包不丢失并且有序到达,TCP 需要对收发两端进行同步确认。首先,客户端向服务器发送同步序列编号(Synchronize Sequence Numbers,SYN)信号来请求同步。服务器端收到 SYN 信号后,发送确认字符(Acknowledge Character,ACK)和 SYN 信号到客户端,客户端收到后再发送 ACK 确认信号向服务器确认。然后再进行 TLS 握手,交换密钥及证书。而对于 QUIC 而言,初次建立连接时,如果客户端没有缓存服务器端的相应配置信息,则需要花费 1 个 RTT 来请求相关信息;如果已缓存了服务器端的相关配置信息,则第二次连接可以依靠缓存的数据直接生成密钥,并立即进行数据传输。这样便能显著减少连接建立的时间,提高数据的传输效率。在对较少数据进行传输及数据传输即时

性要求较高的情况下，QUIC 的 0-RTT 特性降低了建立连接的时间成本。

4.2.2 多路复用解决队头拥塞

多路复用技术最早在 HTTP/2[33] 协议中被提出，它也是 HTTP/2 最经典的设计之一。多路复用使得多个 HTTP 请求在一条 TCP 连接上传输。HTTP/2 多路复用概念的提出解决了 HTTP 层面的队头拥塞[34-36]问题。

如图 4 所示，QUIC 的设计继承了 HTTP/2 的优点，引入了流(stream)的概念。与 HTTP 连接相比，QUIC 可以在同一条连接(connection)上并发地建立多个流，每个流相互独立，不存在约束与依赖关系。即使一个流的数据包丢失，后续流的数据包也能被应用层直接读取并处理。另外，在一条连接中，不同的流之间共享连接信息，所有的流共同参与拥塞控制、丢包检测以及数据加密。

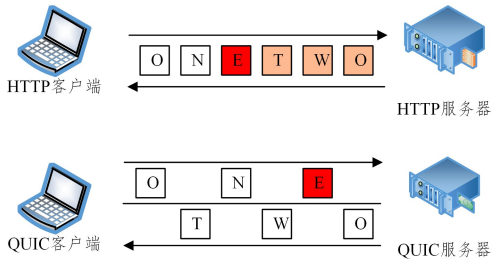



图 4 多路复用

Fig. 4 Multiplexing

4.2.3 解决重传模糊问题

传统 TCP 的拥塞控制算法在 QUIC 中都有相应的实现，如 TCP 默认的拥塞控制算法 Cubic 也是 QUIC 的默认算法。另外，如 Reno，BBR[37] 等拥塞控制算法也在 QUIC 中有相应的实现，但是 QUIC 对 TCP 的重传模糊问题进行了改进。对于 TCP 而言，每一个数据包都有一个独立的序列号(sequence number)。发送端通过观察接收端是否发来 ACK 消息，来确认该数据包是否成功到达。若响应超时或接收端请求重传数据包，则该重传数据包的序列号依旧和原来的相同，导致接收端有时无法辨别到达的数据包是首次发送还是重传发送。而在 QUIC 协议中，数据包序号(packet number)被用来代替 TCP 中的序列号，并且 QUIC 协议保证了每一个数据包序号都严格递增，不会重复。如果当前数据包丢失，则重传数据包的数据包序列号就会在当前发送数据包的数据包序列号之上加 1。同时，QUIC 协议引入流偏移(stream offset)字段来帮助接收端识别该数据包为重传数据包，从而在接收端解决了重传模糊的问题，使接收端对 RTT 能够有更精确的估计[38]。

4.2.4 无缝连接迁移特性

传统 TCP 的连接建立依赖于四元组(源 IP、源端口、目的 IP 和目的端口)来标识一条连接，若四元组中的一项发生改变，连接便会断裂。而 QUIC 协议的连接标识是依赖于连接 ID (connection ID)，而非传统四元组。

如图 5 所示，用户使用蜂窝网络在基站之间进行切换，IP 地址一旦发生变化，TCP 连接便会中断，需要重新建立连接。这一特性使得 TCP 在高速移动的环境下(如高铁环境)的性能表现非常不稳定。而 QUIC 协议的连接并不依赖 IP 和端口的四元组，其通过一个 64 位随机数生成的连接 ID 对连接进行标记。即使在数据传输过程中 IP 和端口都发生了改变，只要客户端和服务器端都记录着相同的连接 ID，那么上层的业务逻辑就能继续保持连接。

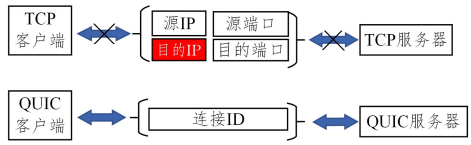



图 5 无缝连接迁移

Fig. 5 Seamless connection migration

### 4.3 QUIC 协议的研究现状

QUIC 相关研究主要分为以下几个方面：QUIC 协议性能测量、QUIC 协议的拓展以及 QUIC 协议的其他方面。

4.3.1 QUIC 协议性能测量

关于 QUIC 协议的性能测试研究，如图 6 所示。我们主要将其归纳为 3 个方面：QUIC 协议文件传输性能、QUIC 协议在流媒体系统中的性能和 QUIC 协议带给用户的体验测试。


图 6 QUIC 协议性能测试部分

Fig. 6 QUIC performance measurement part

在 QUIC 协议的文件传输性能方面，2015 年，Carlucci 等[39] 利用 NetShaper 工具(该工具依托 Netfilter[40] 的 nfqueue 库搭建)控制网络环境(包括带宽与丢包率等网络参数)。并比较了 HTTP，SPDY 和 QUIC 协议的传输网页(web)文件的加载时间，网页中的对象为一些 jpeg 文件。他们通过改变带宽与随机丢包率来模拟不同的网络环境。通过对比 HTTP，SPDY 与 QUIC 协议在各种网络环境下传输网页的加载时间发现：在无随机丢失的网络情况下，QUIC 相比 HTTP 减少了网页的加载时间；在存在随机丢包的网络环境下，QUIC 的性能优于 SPDY。另外，当 QUIC 的前向纠错模块启用后，虽然在接收端可以直接恢复的数据包更多了，但却是以较高的吞吐量为代价换取的，开启 QUIC 自带的前向纠错模块会降低 QUIC 的性能。

2016 年，Biswal 等[41] 将 QUIC 与 HTTP/2 进行性能对比，并在真实环境中使用 QUIC 与 HTTP/2 进行网页的传输。他们发现：1)在比较恶劣的网络环境下，QUIC 协议的性能优于 HTTP/2 的性能，这可能得益于 QUIC 的 0-RTT 连接特性和改进过后的拥塞控制算法。2)当网页中包含一些大的对象时，QUIC 协议的性能会提升；然而当一个网页中包含很多小的对象时，QUIC 相比 HTTP/2 已不再具有优势。Biswal 等认为，增加网页中的对象会影响 QUIC 协议的性能，这是由于低效的浏览器队列(browser queuing)所致。3)在有损网络中 QUIC 协议的性能比 HTTP/2 更好。

2016 年，Megyesi 等[42] 对 QUIC，SPDY，HTTP 协议

性能进行了对比。他们在客户端与服务器之间使用流量控制工具(Traffic Control,TC)进行网络环境的控制,包括网络带宽、往返通信时间和丢包率,并将三者的网页下载时间进行比较。实验中的传输内容为网页,他们对网页中对象的数量和大小进行了控制。在多个实验环境下进行了测试,并得出以下结论:1)当带宽很高(如 50Mb/s),并且有大量的数据需要传输时,QUIC 的性能很差;2)当往返通信时间较长,并且网络带宽较小时,QUIC 的性能显著优于另外两个协议;3)在随机丢包场景中,QUIC 使用前向纠错功能后比不使用前向纠错功能的 HTTP 与 SPDY 的性能好得多;4)小文件传输使用 QUIC 与 SPDY 时,性能优于 HTTP。

2017 年,Cook 等[43]在对 QUIC 进行性能测试时,考虑了网站的资源存放位置并将 QUIC 与 HTTP/2 的网页下载时间进行比较。在实验中,他们对传输的网页进行区分,将传输的网页分为资源存放于同一服务器的网页与资源存放于不同服务器的网页。他们使用 TC 工具对服务器与客户端之间的网络状态进行控制。实验结果表明:在较差的网络环境中,QUIC 的性能优于 HTTP/2;但是在较好的网络环境中,QUIC 的性能优势并不明显。另外他们还提出,网页的开发人员应该考虑网页的资源存放位置,网页资源存放在不同数量的服务器中对网页下载时间具有一定的影响。

2017 年,Zhang 等[44]首次系统性地测试了 QUIC 在卫星网络下的性能,通过使用 TC 工具控制输出队列,模拟卫星网络环境,并将 QUIC、HTTP/2 和基于加密验证层的 HTTP 协议(Hyper Text Transfer Protocol over Secure Socket Layer,HTTPS)在仿真环境下的性能进行测试比较。实验结果表明,QUIC 有助于减少卫星网络下网页页面的加载时间。QUIC 在传输时延较长和丢包率较高的环境下,性能优于 HTTPS 和 HTTP/2。

随后在 2018 年,Wang 等[45]首次在卫星网络环境下测量 QUIC 协议的 BBR 拥塞控制算法,并与 Cubic 拥塞控制算法的性能进行对比。他们利用 TC 工具控制数据队列,模拟卫星网络环境。实验结果表明:1)在吞吐量方面,在卫星网络下,当丢包率为常见的 1%~10%时,BBR 比 Cubic 的吞吐量更高;2)在公平性方面,BBR 的多用户(2 个)平均吞吐量更高,故公平性更好,即在卫星网络下,BBR 拥塞控制算法的性能优于 Cubic 算法。

2017 年,Kakhki 等[46]对多版本的 QUIC 协议在笔记本端、移动端以及有线、无线等环境下进行测量与比较,并与 TCP 协议进行并发请求等测试。主要结论为:1)在笔记本端(desktop)环境下,QUIC 几乎在所有的测试场景中都优于 TCP 结合 HTTPS,这是因为 QUIC 有着 0-RTT 连接、快速恢复丢包等特性。2)在遇到无序数据包的情况下,QUIC 会将这种乱序"理解"为丢包,从而影响协议性能。3)QUIC 依赖于应用层数据包进行处理和加密,这对硬件的计算能力提出了要求。因此,QUIC 在较老型号的手机上性能表现都有所下降。4)在带宽波动的情况下,QUIC 的性能优于 TCP。QUIC 消除了 TCP 的 ACK 歧义性,保证了 QUIC 能更精确地估计 RTT 和带宽。5)QUIC 的公平性不好。当与 TCP 流竞争时,QUIC 占据的带宽是理想平均带宽的两倍。6)QUIC 在流媒体上的性能提升并不大,只有在传输高比特率视频时才具有一定的优势。7)自 2016 年以来,QUIC 的性能改善主要源于扩大了原先保守的最大拥塞窗口。

在 QUIC 协议结合流媒体系统的性能测试方面,2016 年,Timmerer 等[47]对 HTTP/2 与 QUIC 协议结合流媒体系统的性能做了对比。在实验中,客户端以 DASH-JS[48]项目为基础,服务器端为一个标准的 Apache 网页服务器,使用 TC 工具来控制网络环境,通过控制往返通信时间并使网络带宽以固定模式变化来进行实验。将协议开销定义为接收数据总比特数与接收数据中视频数据比特数的差。他们发现,QUIC 协议的实际开销比 HTTP 更大,这可能是加密方式等特性导致的。另外,相比原来的 HTTP 协议,使用 QUIC 协议在流媒体系统客户端吞吐量并不能取得实质性的提升。

2017 年,Bhat 等[49]研究了 QUIC 和 TCP 在基于 HTTP 的动态自适应流媒体(Dynamic Adaptive Streaming over HTTP,DASH)上的性能。他们对比了不同网络环境下,QUIC 和 TCP 中各种自适应算法的 QoE,其中自适应算法包括 BBA[50],SQUAD[51]和 BOLA[52]算法。实验平台的搭建依赖 CloudLab[53],并依赖于红牛数据集[54](RedBull dataset)进行网络环境的设置。他们发现,QUIC 并没有为当前的 DASH 算法带来性能上的提升,相反会带来 QoE 的下降。他们认为,大多数高质量的自适应算法都经过了优化,能够很好地与 TCP 协作,但这些算法并没有充分利用 QUIC 提供的特性,如 ABR 算法基本上不会考虑 QUIC 协议的多路复用特性等。将在 TCP 上成熟的 ABR 算法直接搬移到 QUIC 上,性能反而有所下降。Bhat 等提出了几个基于 QUIC 特性的 DASH 改进思路:1)原本基于 TCP 的自适应码率选择算法可以考虑 QUIC 的多路复用等特性。2)目前所有 DASH 系统的拥塞控制策略包括两个层面,DASH 系统本身与 TCP 传输层层面。而 QUIC 支持使用自定义与标准的拥塞控制协议(如 TCP Cubic,TCP NewReno 以及 BBR)。DASH 客户端可以禁用 QUIC 层面的拥塞控制或者对拥塞控制进行适合流媒体系统结合 QUIC 协议的修改,以提升性能。

快手公司在 2019 年设计了 kQUIC[55]算法,他们结合自身的业务特点,针对短视频场景对 QUIC 做了一系列优化,但技术细节尚未公布。同样针对短视频业务,BIGO 公司对 QUIC 进行改进并取名为 bQUIC[56]。目前 bQUIC 在 BIGO 全球机房都进行了部署,日均承载短视频上传和下载播放近亿次。BIGO 团队仍在对 bQUIC 进行优化,以提升其网络吞吐和抗丢包能力。此外,腾讯公司的腾讯云云计算负载平衡(Cloud Load Balance,CLB)业务目前已经支持使用 QUIC 协议[57-58],这也是国内首家支持 QUIC 协议的云厂商。

在 QUIC 协议的用户体验测试方面,2019 年 Ruth 等[59]研究了 QUIC 的用户体验。针对用户的调研发现,虽然在大多数情况下 QUIC 的性能都优于 TCP,但 TCP 和 QUIC 带给用户的体验的差距较小。他们在修改的 mahimahi[60]框架下

进行实验,调研用户对 QUIC 的体验(QoE),用户被要求回答网页加载过程中是否有协议的切换,并对网页加载速度进行打分。实验结果表明,在网络环境足够好的情况下,用户并不会觉察到两种协议的明显区别。而当网络环境中存在延时或丢包时,QUIC 协议的优势会体验出来,其对于"尾端糟糕体验"(long-tail of bad experiences)有很大的提升,更多的用户可以觉察出两个协议的性能区别。然而总的来说,在较慢的网络中,人们才倾向于使用 QUIC,其对于用户的体验提升并不是非常明显。

2018 年,Marx 等[61]为了增强 QUIC 协议的可测试性(testability)和可调试性(debuggability),提出了 QUIC 通用日志格式的第一个版本,称为 qlog。qlog 可以帮助用户更好地了解 QUIC 协议的运行状态。他们开发了 QUICvis 软件工具集[62](toolset),将运行状态可视化,显示状态包括时间线图(timeline)、序列图(sequence diagram)和拥塞/流控制图(congestion/flow control graph)。

2017 年,Yu 等[63]在仿真环境和真实互联网环境下对 QUIC 和 TCP 进行了测试。该测试通过控制带宽、延迟、缓冲区大小及丢包率来改变网络环境。实验结果表明,在低丢包率的网络中,QUIC 单独使用时具有与 TCP 相当的吞吐量,但和 TCP 一起使用时却缺乏竞争力。QUIC 在高丢包率或较小缓冲区的网络中和 TCP 竞争时,吞吐量大于 TCP。QUIC 在延时较大的网络中时,性能显著下降。文献[64]最后指出,现阶段 QUIC 与 TCP 相比,实用的优势并不大。QUIC 的主要优点在于流的多路复用机制。

#### 4.3.2 QUIC 协议的拓展

在多径 TCP(MultiPath TCP,MPTCP)的启示下,2017 年 Coninck 等[64]率先设计了多路径 QUIC(MultiPath QUIC,MPQUIC)作为 QUIC 的扩展。项目的基础为开源代码 quic-go[65]。他们对每个流下的每个包的公共包头添加了路径 ID(Path ID),用以描述该数据包在传输时对应的不同物理网络路径。在 Mininet 仿真环境下的实验结果表明,多径传输在大多数情况下能改善 QUIC 的性能。此外,对于传输大文件而言,无论是在高延迟/低延迟,还是有丢包/无丢包的情况下,MPQUIC 相比 MPTCP 都有更好的性能表现。对于传输小文件而言,多径传输的表现并不好,这是因为使用多径传输需要多一次握手,使得传输时间有所延长。但如果初始连接建立在一条网络环境较好的链路上,多径的增益还能有所体现。

2018 年,Viernickel 等[66]对 MPQUIC 进行了真实环境和 Mininet 仿真环境下的部署和性能测试。实际测试时,两条网络链路分别为 WIFI 和 LTE。LTE 的数据接收是通过一台 Nexus 5 移动设备连接笔记本电脑的 USB 接口实现的。他们用不同大小的文件和网页对 MPQUIC,MPTCP,QUIC 以及 TCP 进行了性能测试。结果显示,MPQUIC 的性能优于 QUIC,TCP 和 MPTCP。

2018 年,Qian 等[67]提出了一种自适应地使用多路径连接的 mQUIC 方案。他们认为,QUIC 虽然改善了 TCP 的队头阻塞问题,但是单径的 QUIC 在移动网络中仍然会因 RTT 波动以及数据损耗而造成拥塞窗口增长缓慢。而 mQUIC 可以自适应地开启多路径 QUIC 连接,并且检测网络波动,以及时调整拥塞窗口。实验结果显示,无论是丢失感知还是速率感知的拥塞控制算法,这种方案都对数据传输速度均有较大的提升。

2019 年,Coninck 等[68]针对 QUIC 协议的可拓展性展开研究,提出了插件化 QUIC (Pluginized QUIC,PQUIC)框架,该框架允许 QUIC 客户端和服务器在每个连接的基础上,动态地使用协议的拓展插件,如多路径连接或前向纠错等拓展。他们的系统实现了对 QUIC 拓展插件的动态加载,而不用在建立连接时,依赖握手来确定插件的拓展。这种安全且可扩展的技术,使协议插件能够通过 QUIC 连接按需交换。PQUIC 的实现基础为基于 C 语言的 picoquic[69],这是 IETF QUIC 最完善的 QUIC 协议的实现版本之一。此外,他们在 quic-go 中也对 PQUIC 进行了实现[70]。PQUIC 系统实现的动态插入 QUIC 拓展插件包括多路径 QUIC 拓展[64]、不可靠的数据传输拓展[71]和一些前向纠错拓展。测试结果表明,PQUIC 在可接受的开销下,实现了在 QUIC 协议实时运行环境下(protocol runtime environment)拓展插件的动态插入。

#### 4.3.3 QUIC 协议的其他方面研究

2018 年,Hussein 等[72]基于 QUIC 实现了一种增强的软件定义网络(Software Defined Networking,SDN)。在传输大量数据时,可以在更高的带宽利用率下提高安全性。由于当前 QUIC 存在着在高带宽、高流量条件下性能较差、安全性不足、FEC 机制在可变网络下性能较差等问题,因此引入 QUIC 感知的 SDN 架构。通过 SDN 架构可以有效地均衡网络请求,测试目标地址的支持协议,以更好地减小网络延迟以及增强安全性。

2018 年,Wang 等[73]通过将 QUIC 协议部署到内核中,对 QUIC 和 TCP 进行了性能比较。该测试在虚拟机和定制的 WIFI 测试环境(testbed)中完成。结果表明,QUIC 性能在大多数情况下都优于 TCP,并且在并行运行 QUIC 协议时,它们可以公平地共享所有的带宽。

2017 年,Cui 等[74]的综述文章介绍了 QUIC 的相关特性和所面临的挑战,并且讨论了 QUIC 初次建立的情况,0-RTT 连接、多路复用等特性。对于 QUIC 的发展和挑战,文章指出:QUIC 有着比 TCP 更精确的信息反馈机制,可以帮助区分数据包在连接中是丢失还是发生错误,从而有助于改进拥塞控制算法性能。FEC 需要额外的编解码时间,导致额外的能耗,使用时需要慎重考虑。

2020 年,Li 等[75]完成了 QUIC 的综述。他们对 QUIC 协议从总体上做了一个比较详细的介绍。

另外,关于 QUIC 协议的安全性研究与分析也有很多,如文献[76-79]等。由于篇幅有限,且它们并与本文内容的关系不大,便不再展开。

### 4.4 QUIC 协议结合前向纠错编码的发展现状

谷歌曾经在较早版本的 QUIC 协议中尝试使用前向纠错拓展,主要通过在分组中简单地添加一个 XOR 冗余数据包

来实现[80]。在实际测试过程中,FEC 虽然能降低重传率,但是在流媒体服务和网页服务中效用却并不明显,前向纠错编码反而给使用带宽造成了压力,因此最终谷歌公司在 QUIC 中舍弃了 FEC[81]。这是因为实验场景没有充分考虑高铁、无人机等延迟较大、丢包率较高的情况。目前,针对 QUIC 与 FEC 结合应用的研究依然在进行,继 XOR,RS 码后,IETF 也给出了在 QUIC 协议中添加 RLC 冗余编码的执行方案。

2019 年,Michel 等[82-83]为了研究 QUIC 可靠传输机制,自行实现并评估了 QUIC 内的前向纠错拓展。他们在开源项目 quic-go 的基础上定义了一个 FEC 帧(frame),可支持 XOR,RS 码与 RLC 编码。实验在 Mininet 仿真环境下进行,丢包模型调用的是 Gilbert-Elliott 模型。为了避免基于丢包(loss-bassed)策略的拥塞控制信号错误,实验对收到的包(received packets)和恢复的包(recovered packets)做了区分。实验中,他们根据 IETF 标准[84]在 quic-go 项目的基础上增加了基于数据包(packet-based)的检测机制,对不同的 FEC 策略、不同大小的文件等参量进行控制,性能评估的指标为下载完成时间 (Download Complete Time, DCT) 。实验结果证明,当传输大文件或网络环境中延迟较小且丢包率较低时,FEC 冗余编码会对下载完成时间产生负面影响;当传输小文件或是网络环境中延迟较大且丢包率较高时,FEC 冗余编码由于避免了等待超时重传,因此可以极大程度地缩减文件下载时间。

2019 年,Garrido 等[85]提出了 rQUIC,这是首次在 QUIC 中集成了 FEC 功能的流程框架 (framework),并且实现了一种自适应 XOR 的 FEC 算法。其策略为:记录一段时间 $T$ 内总的传输成功的数据包和重传的数据包数量,并且计算剩余丢包率(residual loss)。剩余丢包率表示为:重传的数据包个数与传输的数据包总数减去重传数据包的个数差的商。最后对 $N$ 个时间段内剩余丢包率取平均值,记为平均剩余丢包率。如果剩余丢包率超过一个目标值 $\gamma$,则冗余比增加 $\delta$。实验在 NS3 仿真环境下进行,模拟了低时延(25 ms)、高带宽(20 Mbps)的 WIFI/LTE 网络以及中等时延(100 ms)、中等带宽(10 Mbps)的 2G/3G 网络。在实验中改变随机丢包率,设置每次记录的时间间隔(period $T$)为 3 个 RTT,参数 $\delta=0.33$,$\gamma=1\%$,并将 rQUIC 与 QUIC 传输 20MB 的大文件与 1776KB 的网站(www.flickr.com)的完成时间进行对比。实验结果表明,在存在随机丢包的网络环境中,rQUIC 在传输大文件时至多可以减少 60% 的传输时间。对于传输小文件,rQUIC 的性能依旧优于 QUIC 的性能。

Palmer 等[86]于 2018 年研究了 QUIC 在视频流媒体上的结合应用,并提出了 ClipSream 系统。文章指出,使用 QUIC 协议将流媒体系统中所有视频数据进行可靠传输是一种低效的做法,这样做会使 QUIC 协议在流媒体系统上的表现不好。Palmer 等依据视频帧对 QoE 的影响程度,将其分为重要帧(I 帧)和不重要帧(B 帧和 P 帧)。系统对前者提供可靠传输,对后者提供不可靠传输的同时加上 FEC 冗余。为了支持不可靠传输,系统修改了发送端的发送机制,发送端忽略重传请求而持续发送新数据。测试结果表明,Palmer 等提出的系统在丢包环境下拥有比 TCP 更好的 QoE 表现。

**结束语** 前向纠错编码作为一种冗余编码机制,可以在网络丢包的情况下,依赖收到的原始数据包及冗余数据包恢复丢包数据,避免超时重传。大量研究表明,在存在丢包与高延迟的网络环境下,若能恰当地控制前向纠错算法的冗余度,则将极大地改善数据传输的实时性。QUIC 是谷歌公司提出的基于 UDP 的全新网络协议。QUIC 本身的 0-RTT 连接、多路复用、解决重传模糊等性质,使其在许多场景下都有着优于 TCP 的传输表现。虽然较早版本的 QUIC 选择放弃使用前向纠错编码模块,但这是基于谷歌应用的部分场景作出的判断。在较恶劣的网络环境下,QUIC 和前向纠错编码相结合,性能仍有提升的空间,而基于自适应冗余度控制的前向纠错算法将是 QUIC 协议与前向纠错编码结合的主要发展前景。虽然现已有一些自适应冗余度控制的算法,但得到广泛认可和实际应用的屈指可数。要改变这种情况,需要我们对各种复杂网络状态有更精准的估计,而这方面的研究也是目前的挑战所在。

## 参 考 文 献


[1] CHROMIUM B. Experimenting with QUIC [EB/OL]. (2013-06-27). https://blog.chromium.org/2013/06/experimenting-with-quic.html.

[2] BARAKAT C, ALTMAN E. Bandwidth tradeoff between TCP and link-level FEC[J]. Computer Networks, 2002, 39(2): 133-150.

[3] ALEXIOU A, BOURAS C, KOKKINOS V, et al. Adopting FEC for reliable multicasting over LTE networks[C]// Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems. 2010: 307-310.

[4] WICKER S B, BHARGAVA V K. Reed-Solomon Codes and Their Applications[M]. Piscatway: IEEE Press, 1994: 1-336.

[5] SUDAN M. Decoding of Reed Solomon codes beyond the error-correction bound[J]. Journal of Complexity, 1997, 13(1): 180-193.

[6] ROCA V, CUNCHE M, LACAN J, et al. Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME [EB/OL]. (2013-06-17). https://hal.inria.fr/ hal00799727.

[7] VITERBI A J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm[J]. IEEE Trans. Informat. Theory, 1967, 13(2): 260-269.

[8] ROCA V, SWETT I, MONTPETIT M J. Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC[EB/OL]. (2018-12-12). https://hal.inria.fr/hal-01919876/document.

[9] ROCA V, TEIBI B, BURDINAT C, et al. Less latency and better protection with al-fec sliding window codes: A robust multimedia cbr broadcast case study [C]// 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networ-

king and Communications (WiMob). IEEE, 2017: 1-8.

[10] PADHYE C, CHRISTENSEN K J, MORENO W. A new adaptive FEC loss control algorithm for voice over IP applications [C]// IEEE 2000 International Conference on Performance, Computing, and Communications (Cat. No. 00C-H37086). IEEE, 2000: 307-313.

[11] BALDANTONI L, LUNDQVIST H, KARLSSON G. Adaptive end-to-end FEC for improving TCP performance over wireless links[C]// 2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH3-7577). IEEE, 2004: 4023-4027.

[12] MATHIS M, SEMKE J, MAHDAVI J, et al. The macroscopic behavior of the TCP congestion avoidance algorithm[J]. ACM SIGCOMM Computer Communication Review, 1997, 27(3): 67-82.

[13] LUNDQVIST H, KARLSSON G. TCP with end-to-end FEC [C]// International Zurich Seminar on Communications, 2004. IEEE, 2004: 152-155.

[14] AHN J S, HONG S W, HEIDEMANN J. An adaptive FEC code control algorithm for mobile wireless sensor networks[J]. Journal of Communications and Networks, 2005, 7(4): 489-498.

[15] POLYDOROS A, PAVLIDOU F N, ASSIMAK-OPOULOS C, et al. A Recursive IR Protocol for Multicarrier Communication [EB/OL]. (2020-01-22). http://195. 251. 240. 227/jspui/handle/123456789/4592.

[16] TSUGAWA T, FUJITA N, HAMA T, et al. TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee [C]// Packet Video 2007. IEEE, 2007: 294-301.

[17] CONVERTINO G, OLIVA S L. Adaptive FEC for 802. 11 burst losses reduction[C]// Proceedings of the 2nd International Conference on Mobile Multimedia Communications. 2006: 1-5.

[18] FLACH T, DUKKIPATI N, CHENG Y, et al. TCP Instant Recovery: Incorporating Forward Error Correction in TCP draft-flach-tcpm-fec-00[EB/OL]. (2015-10-14). https://datatracker.ie-tf. org doc/draft-flach-tcpm-fec.

[19] NAGY M, SINGH V, OTT J, et al. Congestion control using fec for conversational multimedia communication[C]// Proceedings of the 5th ACM Multimedia Systems Conference. 2014: 191-202.

[20] FERLIN S, KUCERA S, CLAUSSEN H, et al. Mptcp meets fec: supporting latency-sensitive applications over heterogeneous networks[J]. IEEE/ACM Transactions on Networking, 2018, 26(5): 2005-2018.

[21] CHENG S, HU H, ZHANG X, et al. Deeprs: Deep-learning based network-adaptive fec for real-time video communications [C]// 2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2020: 1-5.

[22] HOCHREITER S, SCHMIDHUBER J. Long short term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.

[23] FU S, ZHANG Y, JIANG Y, et al. Experimental study for multi-layer parameter configuration of WSN links [C]// 2015 IEEE 35th International Conference on Distributed Computing Systems. IEEE, 2015: 369-378.

[24] LOHMAR T, PENG Z, MHNEN P. Performance evaluation of a file repair procedure based on a combination of MBMS and unicast bearers[C]// International Symposium on World of Wireless, Mobile & Multimedia Networks. IEEE, 2006.

[25] LUBY M, SHOKROLLAHI A, WATSON M, et al. Raptor forward error correction scheme for object delivery[R]. RFC 5053 (Proposed Standard), 2007.

[26] SHOKROLLAHI A. Raptor codes [J]. IEEE Transactions on Informon Theory, 2006, 52(6): 2551-2567.

[27] ALEXIOU A, BOURAS C, KOKKINOS V, et al. Adopting FEC for reliable multicasting over LTE networks [C]// Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems. 2010: 307-310.

[28] BOURAS C, KANAKIS N, KOKKINOS V, et al. Evaluating RaptorQ FEC over 3GPP multicast services[C]// 2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2012: 257-262.

[29] PIRO G, GRIECO L A, BOGGIA G, et al. Simulating LTE cellular systems: An open-source framework[J]. IEEE Transactions on Vehicular Technology, 2010, 60(2): 498-513.

[30] ZHONG Z, HAMCHAOUI I, KHATOUN R, et al. Performance evaluation of CQIC and TCP BBR in mobile network[C]// 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). IEEE, 2018: 1-5.

[31] RUTH J, POESE I, DIETZEL C, et al. A First Look at QUIC in the Wild[C]// International Conference on Passive and Active Network Measurement. Cham: Springer, 2018: 255-268.

[32] BISHOP M. Hypertext Transfer Protocol Version 3 (HTTP/3) [EB/OL]. (2020-12-15). https://tools. ietf. org/html/draft-ietf-quic-http-33.

[33] BELSHE M, THOMSON M, PEON R. Hypertext Transfer Protocol Version 2 (HTTP/2) [EB/OL]. (2015-05-02). https://tools. ietf. org/html/rfc-7540.

[34] SCHARF M, KIESEL S. NXG03-5: Head-of-line Blocking in TCP and SCTP: Analysis and Measurements[C]// IEEE Globecom 2006. IEEE, 2006: 1-5.

[35] QIAN F, GOPALAKRISHNAN V, HALEPOVIC E, et al. Tm3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking [C]// Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. 2015: 1-13.

[36] MI X, QIAN F, WANG X. Smig: Stream migration extension for http/2[C]// Proceedings of the 12th International on Conference on Emerging Networking Experiments and Technologies. 2016: 121-128.

[37] CARDWELL N, CHENG Y, GUNN C S, et al. BBR: congestion-based congestion control [J]. Communications of the ACM, 2017, 60(2): 58-66.

[38] IYENGAR J, THOMSON M. QUIC: A UDP-Based Multiplexed and Secure Transport [EB/OL]. (2020-06-10). https://datatracker. ietf. org/doc/html/draft-ietf-quic-transport-29.

[39] CARLUCCI G, CICCO L D, MASCOLO S. HTTP over UDP: an Experimental Investigation of QUIC[C]// Proceedings of the 30th Annual ACM Symposium on Applied Computing. 2015: 609-614.

[40] NETFILTER'S W. The netfilter. org project[EB/OL]. (2021-01-15). http://www. net-filter. org/.

[41] BISWAL P, GNAWALI O. Does quic make the web faster? [C]// 2016 IEEE Global Communications Conference (GLOBECOM). IEEE, 2016: 1-6.

[42] MEGYESI P, KRAMER Z, MOLNAR S. How quick is QUIC? [C]// 2016 IEEE International Conference on Communications (ICC). IEEE, 2016: 1-6.

[43] COOK S, MATHIEU B, TRUONG P, et al. QUIC: Better for what and for whom? [C]// 2017 IEEE International Conference on Communications (ICC). IEEE, 2017: 1-6.

[44] ZHANG H, WANG T, TU Y, et al. How quick is quic in satellite networks[C] // International Conference in Communications, Signal Processing, and Systems. Singapore: Springer, 2017: 387-394.

[45] WANG Y, ZHAO K, LI W, et al. Performance evaluation of QUIC with BBR in satellite internet[C]// 2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE). IEEE, 2018: 195-199.

[46] KAKHKI A M, JERO S, CHOFFNES D, et al. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols[C]// Proceedings of the 2017 Internet Measurement Conference. 2017: 290-303.

[47] TIMMERER C, BERTONI A. Advanced transport options for the dynamic adaptive streaming over HTTP [J]. arXiv: 1606. 00264, 2016.

[48] ANGOT N. Dash Industry Forum [EB/OL]. https://github. com/Dash-Industry-Forum/dash. js.

[49] BHAT D, RIZK A, ZINK M. Not so QUIC: A performance study of DASH over QUIC[C]// Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video. 2017: 13-18.

[50] HUANG T Y, JOHARI R, MCKEOWN N, et al. A buffer-based approach to rate adaptation: evidence from a large video streaming service[C]// Proceedings of the 2014 ACM Conference on SIGCOMM. 2014: 187-198.

[51] WANG C, RIZK A, ZINK M. SQUAD: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP[C]// Proceedings of the 7th International Conference on Multimedia Systems. 2016: 1-12.

[52] SPITERI K, URGAONKAR R, SITARAMAN R K. BOLA: Near-optimal bitrate adaptation for online videos[J]. IEEE/ACM Transactions on Networking, 2020, 28(4): 1698-1711.

[53] RICCI R, EIDE E, TEAM C L. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications [J]. Magazine of USENIX & SAGE, 2014, 39(6): 36-38.

[54] LEDERER S, MULLER C, TIMMERER C. Dynamic adaptive streaming over HTTP dataset[C]// Proceedings of the 3rd Multimedia Systems Conference. 2012: 89-94.

[55] TECHNOLOGY K. Kuaishou Technology developed kQUIC by itself: How to realize tens of millions of QPS clusters? [EB/OL]. (2020-07-13). https://new. qq. com/rain/a/20200730A0FGC-W00.

[56] BIGO. BIGO Technology|The practice and optimization of QUIC protocol in BIGO [EB/OL]. (2020-12-15). https://www. nowcoder. com/discuss/579715.

[57] TENCENT. Using QUIC Protocol on CLB[EB/OL]. (2020-08-15). https://intl. cloud. tencent. com/doc-ument/product/214/37353?lang=en.

[58] CLB TEAM. CLB quic demo[EB/OL]. (2017-11-9). https://github. com/tencentyun/clb-quic-demo.

[59] RUTH J, WOLSING K, WEHRLE K, et al. Perceiving QUIC: Do users notice or even care? [C]// Proceedings of the 15th International Conference on Emerging Networking Experiments and Technologies. 2019: 144-150.

[60] NETRAVALI R, SIVARAMAN A, DAS S, et al. Mahimahi: Accurate record-and-replay for {HTTP} [C]// 2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15). 2015: 417-429.

[61] MARX R, LAMOTTE W, REYNDERS J, et al. Towards QUIC debuggability[C]// Proceedings of the Workshop on the Evolution, Performance, and Interrability of QUIC. 2018: 1-7.

[62] MARX R. QUICvis [EB/OL]. (2019-10-02). https://github. com/rmarx /QUICvis.

[63] YU Y, XU M, YANG Y. When QUIC meets TCP: An experimental study[C]// 2017 IEEE 36th Internatinal Performance Computing and Communications Conference (IPCCC). IEEE, 2017: 1-8.

[64] CONINCK Q D, BONAVENTURE O. Multipath quic: Design and evaluation[C]// Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies. 2017: 160-166.

[65] SEEMANN M. A QUIC implementation in pure Go[EB/OL]. (2021-05-01). https://github. com/lucas-clemente/quic-go.

[66] VIERNICKEL T, FROEMMGEN A, RIZK A, et al. Multipath QUIC: A deployable multipath transport protocol[C] // 2018 IEEE International Conference on Communications (ICC). IEEE, 2018: 1-7.

[67] QIAN P, WANG N, TAFAZOLLI R. Achieving robust mobile web content delivery performance based on multiple coordinated QUIC connections[J]. IEEE Access, 2018, 6: 11313-11328.

[68] CONINCK Q D, MICHEL F, PIRAUX M, et al. Pluginizing quic [C]// Proceedings of the ACM Special Interest Group on Data Communication. 2019: 59-74.

[69] HUITEMA C. picoquic [EB/OL]. (2021-04-03). https://github. com/ private-octopus/picoquic.

[70] CONINCK Q D, BONAVENTURE O. The Case for Protocol Plugins[R]. Technical Report, 2019.

[71] KINNEAR E, PAULY T, SCHINAZI D. An Unreliable Data-

gram Extension to QUIC[EB/OL]. (2021-04-21). https://quic-wg.org/ datagram/draft-ietf-quic-datagram.html.

[72] HUSSEIN A,KAYSSI A,ELHAJJ I H,et al. SDN for QUIC: An enhanced architecture with improved connection establishment[C]// Proceedings of the 33rd Annual ACM Symposium on Applied Computing. 2018:2136-2139.

[73] WANG P,BIANCO C,RIIHIJARVI J,et al. Implementation and performance evaluation of the quic protocol in linux kernel [C]// Proceedings of the 21st ACM International Conference on Modeling,Analysis and Simulation of Wireless and Mobile Systems. 2018:227-234.

[74] CUI Y,LI T,LIU C,et al. Innovating transport with QUIC:Design approaches and research challenges[J]. IEEE Internet Computing,2017,21(2):72-76.

[75] LI X B,CHEN Y,ZHOU M Y,et al. Internet Data Transfer Protocol QUIC:A Survey[J]. Journal of Computer Research and Development,2020,57 (9):1864-1876.

[76] FISCHLIN M,GUNTHER F. Multi-stage key exchange and the case of Google's QUIC protocol[C]// Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014:1193-1204.

[77] LYCHEV R,JERO S,BOLDYREVA A,et al. How secure and quick is QUIC? Provable security and performance analyses [C]// 2015 IEEE Symposium on Security and Privacy. IEEE, 2015:214-231.

[78] JAGER T,SCHWENK J,SOMOROVSKY J. On the security of TLS 1.3 and QUIC against weaknesses in PKCS# 1 v1.5 encryption[C]// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015:1185-1196.

[79] AVIRAM N,SCHINZEL S,SOMOROVSKY J,et al. {DROWN}:Breaking {TLS} Using SSLv2[C]// 25th {USNIX} Security Symposium ({USENIX} Security 16). 2016:689-706.

[80] HAMILTON R. QUIC:A UDP-Based Secure and Reliable Transport for HTTP/2[EB/OL]. (2016-01-13). https://tools.ietf.org /html/draft-tsvwg-quic-protocol-02.

[81] LANGLEY A,RIDDOCH A,WILK A,et al. The quic transport protocol:Design and internet-scale deployment[C]// Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017:183-196.

[82] MICHEL F,CONINCK Q D,BONAVENTURE O. QUIC-FEC:Bringing the benefits of Forward Erasure Correction to QUIC[C]// 2019 IFIP Networking Conference (IFIP Networking). IEEE,2019:1-9.

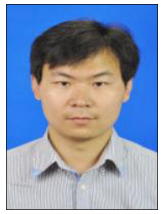
[83] MICHEL F,CONINCK Q D,BONAVENTURE O. Adding forward erasure correction to quic[J]. arXiv:1809.04822,2018.

[84] IYENGAR J,SWETT I. QUIC Loss Detection and Congestion Control[EB/OL]. (2021-04-16). https://datatracker.ietf.org/doc/html /draft-ietf-quic-recovery-32.

[85] GARRIDO P,SANCHEZ I,FERLIN S,et al. rQUIC:Integrating FEC with QUIC for robust wireless communications[C]// 2019 IEEE Global Communications Conference(GLOBECOM). IEEE,2019:1-7.

[86] PALMER M,KRUGER T,CHANDRASEKARAN B,et al. The quic fix for optimal video streaming[C]// Proceedings of the Workshop on the Evolution,Performance,and Interoperability of QUIC. 2018:43-49.

**LIN Li-xiang**, born in 1996, postgraduate. His main research interests include network transmission protocol, streaming media and so on.



**XU Yue-dong**, born in 1980, Ph.D, associate professor. His main research interests include computer network, distributed machine learning and so on.

(责任编辑:喻藜)